Open-Apple™

Releasing the power to everyone.

July 1985
Vol. 1, No. 6

Sculley reorganizes Apple, Inc.

Apple's Macintosh division finally got its balloon punctured May 31, when John Scully, Apple's president, announced a reorganization of the company. Two weeks later, on June 14, Apple announced it was moving the production of the Apple IIc to its "Macintosh" plant in Fremont, Calif; moving production of all IIes to its factory in Singapore; and firing 21 per cent of its work force. Sixty per cent of the firings are among manufacturing workers. All Apple factories, except the Fremont and Singapore plants and one in Cork, Ireland, will be closed.

Apple's board of directors decided to allow Sculley to reorganize the company from two *product*-oriented divisions (Apple II and Macintosh) into two *function*-oriented divisions (operations and marketing). In the shuffle Steve Jobs, founding father and chairman of Apple, lost his operating responsibilities, (previously he had direct control over both the manufacturing and marketing of the Macintosh). He remains chairman of the company.

Del Yocam, the Apple veteran who had been in charge of the Apple II division, is now head of all operations, including manufacturing, distribution, and new product development. Yocam, 41 years old, joined Apple in 1979. Yocam's accomplishments at Apple include coordinating the manufacturing changeover from the Apple II-Plus to the Apple IIe in late 1982, and overseeing the development and introduction of the Apple IIc. In a *Wall Street Journal* article on Yocam and Apple (June 7, page 18), Steve Wozniak said that during Yocam's tenure, "the direction of the Apple II division is the best it has been in five years."

Jean-Louis Gassee will work for Yocam as head of product operations, which includes new product development. In 1980, Gassee formed an Apple distributorship in France. The distributorship later became Apple France, a company subsidiary. Under Gassee, Apple France captured about 30 per cent of the French personal computer market; more than IBM and more than Apple's share in other countries. In its June 10 edition, *InfoWorld* (page 15; caution, this article was written before Sculley's reorganization and has Gassee's position wrong) describes Gassee as an intellectual and a visionary —a more mature French counterpart to Steve Jobs. However, he is also said to be an experienced turnaround manager and to have an ability to satisfy dealers.

Apple's new marketing division will be headed by William Campbell, who was recruited by Sculley in late 1983 from Eastman Kodak. At Kodak, Campbell was director of consumer product marketing in Europe. At Apple, he had been directly associated with neither the Apple II nor Macintosh divisions, but was the top marketing person on Apple's corporate staff.

Sculley's reorganization is the result of a leveling-off in the growth rate of personal computer sales. Apple's sales are actually higher than last year, however, they are not as high as planned. (In 1984 Apple's revenues were 54 per cent higher than the previous year. 1985's growth in sales will be considerably less.) Because of costs associated with the reorganization, Apple expects to post its first quarterly loss at the end of June.

The slowdown in the growth rate of computer sales is affecting the entire industry. The usual profit growth Wall Street has become accustomed to just isn't materializing this year. IBM has announced that it expects no change in profits during the first nine months of 1985 compared to the previous year. Apple expects to earn a lower profit this year than last. Wang, for the first time in its history, is predicting it will earn no profit at all.

The slowdown has hurt Apple more than IBM because the situation is worse in the consumer market, where Apple is strongest, than in the business market, where IBM leads. Much like just before the IBM PC jr appeared on the scene, many consumers are delaying computer purchases until new, Macintosh-like computers announced by Commodore and Atari are on dealers' shelves. Look for Apple II sales to surge—and shortages to develop—when those machines finally hit the streets, just as happened when the PC jr came out of IBM's closet.

My Two Bits

by Tom Weishaar

Apple hasn't asked for my advice (merely two subscriptions have been paid for with checks from Cupertino...) but let's take a look at the Apple II family and send a few thoughts about the future back to California anyhow.

For six years now Apple's management has been trying to build a computer better than the Apple II. During those six years the scorned II has provided Apple with millions of dollars for research and development. The money was spent on the Apple III and Lisa, both now dead, and the Macintosh, which isn't dead yet. In all these years, after all this money, only the Apple II has ever shown a profit.

The message here is that computer buyers are not looking for the fastest chips, the biggest memory, the best graphics, or the largest disk drives. They aren't looking for "user-friendliness" that sacrifices programmability and speed for mice and bit-mapped trashcans.

Buyers are looking for tools they can use. The Apple II has plenty of power to be useful. So have several hundred other machines.

The significant thing about the Apple II is the community of people that has sprung up around the machine, teaching other people how to use it, developing applications for it, designing hard and software for it. *It is this community that makes the Apple II a significant machine. It is this community that is the driving force behind Apple Inc's phenomenal growth.*

The fastest road to glory for Apple is to continue a slow evolution in the Apple II family. The fastest road to ruin is to develop machines that abandon this community in Apple's technological wake.

The Apple II community is averse to walking away from its hard-earned understanding of the Apple II. If the community can't leverage its accumulated knowledge into better explanations for new users, better applications, and better hard and software, a new machine will not get its interest. If we are forced to walk away from the Apple II, we are as likely to walk in the direction of IBM or Atari as in whatever direction Apple Inc. is "leading."

A slow evolution would allow features to be added to new high-end models. But the compatibility of these new models with previous models is a far more important issue than the latest technological wizardry.

Compatibility means a microprocessor based on the 6502, Applesoft in ROM, Apple II-compatible memory mapping, Apple II-compatible slots, and 5-1/4 inch disk drives as standard equipment. It means computers that will start up DOS 3.3 disks with no special handling. And it means reasonable prices. Any computer that has these features will be a success in the marketplace.

From its base in education, the home, and small business, the Apple II's market could be gradually broadened to include larger organizations by the introduction of more powerful machines and software. The technology is available to build a family of computers around the Apple II.

At the top could be a true 16-bit computer based on the 65816 microprocessor. It should have a ton of memory. But it should also have some standard Apple II slots. How about a souped-up version of *AppleWorks* to go with this machine, with macros, graphics, terminal-emulation software, and desktop accessories? But be careful, Apple. Unless this machine can also boot and run APPLEVISION, the early Integer Basic hi-res graphics and sound demonstration program, it's not an Apple II.

In the middle could be a "16-bit" computer based on the 65802. The 65802 has internal registers 16 bits wide but reads and writes to and from memory 8 bits at a time (just like the IBM-PC). Because of this, and because (like the 65816) it acts like a 6502 when you turn it on, you can actually plug one of these into an existing Apple II. Imagine a machine with 256K to 1 megabyte of memory built around one of these at the price of today's Apple IIe. But it's gotta run APPLEVISION.

Then there's the IIe and IIc. Gradually falling prices and the inclusion of *AppleWorks* in the box with the computer could extend the lives of these machines beyond anything IBM ever dreamed of. And we already know they run APPLEVISION.

In addition to computers, Apple can look for growth in the peripherals market. With the major exception of the Laserwriter, Apple has shown no technological leadership at all in the area of printers, modems, or monitors. An **add-on** 800K 3-1/2 inch drive, at a good price, could certainly be a successful Apple II peripheral (but its incompatiblity with existing software would crush the sales of any "Apple II" it was built into).

And how about special private label models of the II that could be sold to companies that want to build other things out of them? This is a market Apple has totally ignored. Since starting *Open-Apple*, I've corresponded with people who have Apple IIs at the center of electronic weighing systems, cash registers, and library circulation systems, among others. These people all sell products that happen to include Apple IIs. Because there are so many people in the world who know how to make an Apple II dance, there is a tremendous unexploited market here. It's easy to imagine Apples hidden away at the heart of security systems, engine or blood or fertilizer analyzers, or (dare I say it?) telephones. But Apple has to learn to pursue these "low-tech" Apple II-based markets and stop wasting its resources trying to seduce the Fortune 500 with the Macintosh.

Does Apple's sudden reorganization mean the people who run it have finally realized the potential shyly hiding inside the Apple II? We'll have to wait and see.

Digging Into DOS

The /RAM disk that ProDOS automatically installs on 128K Apples is a feature that can be hard to take advantage of. If you have a very big program, you can theoretically split the program into sections, put the sections on /RAM, and use the CHAIN command to quickly move from one section to another. Or, if you have a small program but a large data file, you can theoretically put the data file on /RAM and access it at the speed of light instead of your disk drive's 300 RPM.

To change these ideas from theory to practice, however, requires a program that will copy files from disk to /RAM when you start up your computer, and from /RAM back to disk just before you turn it off. You *can* use the ProDOS *FILER* for this, but it doesn't give the kind of automatic operation most computer users are looking for.

Several subscribers have written asking for a Basic program that can copy files from disk to /RAM and back. Such a program turns out to be extremely simple to write because of some powerful parameters Basic.system provides for binary file commands. Since many people haven't discovered this power yet, let's examine these parameters and write a basic Basic copy program for ProDOS.

ProDOS and Basic.system version numbers. Before we start, let's talk about the various versions of ProDOS, since there are some version differences in how the binary commands work. Before we can even talk about versions, however, let's talk about something even more elementary—the difference between the ProDOS kernel and Basic.system. A major fault with the ProDOS documentation, beginning with Apple's stuff and extending right through almost everything else that has been written, is a failure to distinguish between the ProDOS kernal and Basic.system.

The ProDOS kernal lives in a file called *PRODOS*. I've found four official versions in my disk library (there may be more); 1.0.1 (1-JAN-84), 1.0.2 (15-FEB-84), 1.1 (17-AUG-84), and 1.1.1 (18-SEP-84). You can tell which version is on a disk by cataloging the disk and looking at the date the file was last *modified* (not *created*, that's when the FILER put it on your disk). You can also tell by booting the disk and watching the screen carefully. The version number of the ProDOS kernel and a copyright notice appear for about 4 seconds while the disk is booting.

ProDOS is accessed through a *machine language interface*. The ProDOS kernel supports no other language, only machine language. It has commands such as *set_file_info, get_eof, and alloc_interrupt*. It doesn't know control-D from synchronous idle. It doesn't even have a CATALOG command.

Basic.system lives in a file called *BASIC.SYSTEM*. It's been officially released in only two versions that I can find; 1.0 (15-NOV-83) and (15-MAR-84) and 1.1 (18-JUN-84). You can tell which version is on a disk by cataloging the disk and checking the date. You can also start up BASIC.SYSTEM in such a way that it can't find a *STARTUP* file (temporarily change the name of the STARTUP file to something else); Basic.system prints its version number after booting if there's nothing to start up.

The commands everyone refers to as "ProDOS" commands, such as CATALOG, BLOAD, and PREFIX, are really "Basic.system" commands. "Basic.system" is the connecting link between Applesoft and the ProDOS kernel. Basic.system provides a DOS 3.3-like control-D interface for Basic programmers working under ProDOS. The likeness to DOS 3.3, however, is *not* a feature of ProDOS; Logo runs under ProDOS and can't even print a control-D.

Binary file anatomy and execution. Binary files typically hold either machine language programs or high-resolution graphics images. They're the ones identified by *BIN* in a disk's catalog.

Binary files can be executed with either the DASH ("-") or the BRUN command, like this:

```
- /MY.DISK/MY.PROGRAM

BRUN /MY.DISK/MY.PROGRAM
```

These commands work best, of course, if the file really is a machine language program. If the file contains a graphic or hides some other type of data, an attempt to execute it will usually make your computer lock up or crash into the Monitor.

Think of binary files as "snapshots" of portions of your computer's memory. They are very similar to the photographic images a Kodak takes. However, they have one significant advantage—you can snap them back into memory the next day and take up where you left off. That's something we can't do yet with my baby pictures.

When you take a snapshot of memory, you have to specify what address range you want to take a picture of. Basic.system provides three parameters for doing this. The *A* parameter (address) specifies where in memory the range starts. You specify where it ends with either the *E* parameter (end), which gives the address of the last byte in the range, or with the *L* parameter (length), which tells how long it is.

Basic.system stows this memory-position information, along with other data about the file, inside the disk's directory. The 80-column ProDOS CATALOG will dig it back out for you. Here's a ProDOS catalog with a file holding a hi-res picture. The address of the file is given (in hex) under "subtype;" the length is given (in decimal) under "endfile."

```
NAME          TYPE  BLOCKS  MODIFIED         CREATED          ENDFILE SUBTYPE

PRETTY.PIC    BIN     17  31-JUN-85  7:47  31-JUN-85  7:47     8192 A=$2000
```

If you use the DASH command to execute a program, Basic.system will use the addresses in the directory to determine where to load the file. The same thing happens with the BRUN command, although with BRUN you can override the "default" addresses with the *A*, *E*, or *L* parameters.

Get a load of this. Binary snapshots can be loaded into your computer without executing them with the command BLOAD. A feature found in Basic.system's BLOAD and BRUN commands that's not in DOS 3.3 is that you

can use the L (or E) parameter to load just part of a binary file. You don't have to load the whole thing. If you use the L parameter with BLOAD or BRUN under DOS 3.3 you get a syntax error.

Another difference from DOS 3.3 is the Basic.system *file type* parameter, T. This parameter can be used with BLOAD (but not with BRUN) to load *any* type of file into memory as if it were a binary snapshot. This is a very powerful feature; without it our soon-to-follow basic Basic copy program would be impossible.

The binary savior. Binary snapshots are created with the command BSAVE. If the file doesn't already exist, you have to specify an address range for the picture. Here's an example:

```
BSAVE PRETTY.PIC,A$2000,L$2000
```

or

```
BSAVE PRETTY.PIC,A8192,E$3FFF
```

When the file doesn't already exist, BSAVE will create a new one and tuck your snapshot into it. An interesting side effect of using BSAVE with the file type parameter, however, is that with T, BSAVE can't create new files. It returns a PATH NOT FOUND error, *even if the file type you specify is BIN.*

If the file does already exist, on the other hand, Basic.system has some more magic for us. Unlike DOS 3.3, Basic.system doesn't make you specify an address range for existing files. *BSAVE PRETTY.PIC* works just fine—Basic.system will use the existing address and length information.

The big difference between Basic.system 1.0 and 1.1 is what happens when the file already exists and you specify a *different* address range. With 1.0, as with DOS 3.3, the old address range is thrown out and the new one is used. If the new range is shorter than the previous one, any unneeded disk blocks are released. With 1.1, on the other hand, the old address range is retained. (If the new material makes the file longer, however, the file length will be increased as required.)

Sandy Mossberg (letter in ***Nibble***, June 1985, page 7) and Glen Bredon (letter in ***Open-Apple***, May 1985, page 40), both competent authorities on such stuff, say this is a bug in Basic.system 1.1.

But Ken Kashmarek and Cecil Fretwell, also competent authorities, argue in letters to ***Open-Apple*** that it's not a bug at all. More about this is a moment.

It's painful to B misunderstood. First, though, let's examine the most misunderstood little creature in all of Appledom, the B (byte) parameter. B got off on the wrong foot in the original DOS 3.3 documention, *The DOS Manual.* That book had a two-page section on the B parameter that began, "Note: the following section is *not* for beginners...", and which consisted mostly of warnings about all the problems you could create by using the B parameter inappropriately.

In the later *DOS Programmer's Manual,* this two-page section is missing. What little information on B that is presented in this book is only partially correct.

In *Basic Programming with ProDOS,* the B parameter loses even its entry in the index. No examples of how to use the parameter are given. And the tradition continues in Gary Little's ProDOS series running in ***A+*** and in Lee Swoboda's series running in ***inCider***—neither of them say much about B and both of them have its maximum value wrong.

The B parameter allows you to quickly and easily move the position-in-file pointer to any byte in a file. Unlike DOS 3.3, Basic.system allows you to use B with BLOAD and BSAVE. B can also be used (under both DOS 3.3 and ProDOS) with READ and WRITE, but this is more complicated; for a complete discussion see the August 1984 *DOStalk* (in ***Softalk***, page 43).

When you use B and L (or E) together, you suddenly have some amazing new binary-file-handling abilities. You can put several graphics in the same file and access them one at a time. You can put several programs in one file, too.

The best part, though, is using B and L (or E) and T together at the same time. This allows you to read and write records to and from any part of a text file at binary-file speeds. For example, imagine a hard-disk based data file with 1,000 records, each 1,200 bytes long. Reading in a 1,200 byte record can take awhile with INPUT; but imagine how long it would take with:

```
PRINT D$;"BLOAD DATA.FILE, TTXT, A$4000, L1200, B";R*1200
```

That command would load the 1,200 byte record specified by R into the memory range beginning at $4000 in a fraction of the time INPUT would take.

But consider for a moment the equivalent command for saving to the data file, and you'll understand why our correspondents Kashmarek and Fretwell don't see a bug where Mossberg and Bredon do:

```
PRINT D$;"BSAVE DATA.FILE, TTXT, A$4000, L1200, B";R*1200
```

If Basic.system's BSAVE works like DOS 3.3's in this situation, all the data in the file beyond the current record gets cut off and thrown away. That's not an acceptable alternative.

But, on the other hand, if BSAVE doesn't adjust the length of binary files, you'll often end up with files that have junk tacked on the end—as frequently happens with text files.

If the ProDOS modification team thinks like I do, someday they'll fix BSAVE so it cuts off the end of the file, but *only if no file type parameter is given.* Treat the T like Mr. T, I say, and don't cut anything from files when a BSAVE uses it.

Xerox will love this. Now that everybody understands the magic of BLOAD and BSAVE under Basic.system, let's write a quick routine for copying files from disk to /RAM and back. We'll BLOAD the files with one prefix, then quickly BSAVE them with another prefix. Just one problem; *where* in memory do we load them? Our program and its variables will be in memory somewhere; it's important that we don't load a file on top of anything.

Refer back to the memory map in our January article "Taking a poke at the Garbageman" (page 4). The map shows that there's an area of free memory between a Basic program's variable tables and its string storage area. The map shows what bytes to peek at to find the boundaries of this free area. And the article explains that the FRE command makes this area as large as possible.

So, to get some memory we'll execute a FRE command; peek into the location known as STREND at 109-110 ($6D-6E) to get an Address parameter; and peek into the location known as FRETOP at 111-112 ($6F-70) and subtract 1 to get an End parameter. How about:

```
 10 REM ***  BASIC.COPY  ***
100 D$=CHR$(4)
110 P1$="/JLY/"  : REM prefix of volume that files are to be copied FROM
120 P2$="/RAM/"  : REM prefix of volume that files are to be copied TO

130 F$="FILE1" : T$="TXT" : GOSUB 400 : REM names and types of
140 F$="FILE2" : T$="BAS" : GOSUB 400 : REM   files to be copied
150 F$="FILE3" : T$="BIN" : GOSUB 400

200 END

400 B=0 : A=0 : E=0 : L=0 : REM Important--messing with this line can be fatal
410 PRINT D$;"FRE"
420 A=PEEK(109) + PEEK(110)*256
425 E=PEEK(111) + PEEK(112)*256

430 ONERR GOTO 500
440 PRINT D$;"CREATE"; P2$;F$; ",T";T$

450 PRINT D$;"BLOAD"; P1$;F$; ",T";T$; ",A";A; ",E";E-1; ",B";B
460 L=PEEK(48859) + PEEK(48860)*256
470 PRINT D$;"BSAVE"; P2$;F$; ",T";T$; ",A";A; ",L";L; ",B";B
480 IF L=(E-A) THEN B=B+L : GOTO 450

490 POKE 216,0 : RETURN

500 IF PEEK(222)=19 THEN PRINT D$;"DELETE"; P2$;F$ : RESUME
510 IF PEEK(222)=5  THEN CALL -3288 : GOTO 490
520 PRINT "ERROR #";PEEK(222);" IN LINE ";PEEK(218) + PEEK(219)*256
530 END
```

see P76

Don't mess with line 400. It makes sure all the variables used in the routine have been allocated space in memory *before* we find out how much free space we have. If a variable is used for the first time *after* we do our peeks to determine A, the variable table will get moved up into our free space with not inconsequential consequences.

Since BSAVE—when used with the T parameter—can't create files, we have to CREATE a file for our /RAM copy in line 440. If the file already exists, (as it most certainly will when we copy from /RAM back to disk), a DUPICATE PATHNAME error will occur and line 430's ONERR will take us to line 500. The error code for duplicate pathname is 19; line 500 deletes the file and RESUMEs back to the CREATE command.

Line 450 BLOADS the file. If the file is *shorter* than what we have specified with the E parameter, no error will occur. If it is *longer,* E sees to it that we load only as much of the file as will fit. Loading begins with the first byte of the file (byte 0).

Line 460 peeks at a Basic.system location known as RWTRANS at 48859-60 ($BEDB-DC) to find out how many bytes actually were loaded by the BLOAD command. (The equivalent location in DOS 3.3 is 43616-17 ($AA60-61); the actual loading Address can be found at 48855-56 ($BED7-D9), this is equivalent to DOS 3.3's 43634-65 ($AA72-73).)

Line 470 BSAVES the file using it's actual Length. This Length will transfer to the new copy of the file correctly. However, if the file really is a binary file, the copy's loading Address, as shown in the catalog, will be wrong. So will the record length (R=) of text files. Basic.system 1.0 shows the address we used in parameter A, not the one on the original file, for both Address and record length. Version 1.1, on the other hand, sets both to zero. (Incidentally, if you try to BLOAD a file with a loading address of zero you get a NO BUFFERS AVAILABLE error. This message is quite confusing, because the problem has nothing to do with buffers; the ProDOS kernel simply refuses to load files into that address because the area is "protected" in the "system bit map.")

These problems are fixable, but not this month. For now, always specify A and L when using files copied by this techinque.

Line 480 looks to see if we got the whole file or if there is more of it still on the disk. If the actual Length we got was less than the End minus the Address, we got it all. If these values are equal, the file was either exactly the same size as our free memory area, or there is more. In either case, adjust the B parameter so it points to the first byte we didn't get, and go back to line 450.

The second time we execute BLOAD, the B parameter points well inside the file. We'll get a bunch more of it; this will continue until we have the whole file. If we should run across a file that accidentally has the same number of bytes as our free space, line 450 will get an END OF DATA error. Line 430's ONERR will then take us to line 510, where we clear the stack (see January's *Digging Into DOS*, page 2), and return to the caller.

There you have it—a basic Basic copy routine.

This one didn't stick. In preparing this article, I took a look at what all the books I have on ProDOS had to say about binary files. This brought me face-to-face with the fact that Chapter 6 of Prentice-Hall's *Inside Apple's ProDOS* is an idea-for-idea rip-off of Chapter 9 of Apple's own *Basic Programming with ProDOS*. For example, compare these two sentences:

Because of the way ProDOS uses memory, it is difficult to predict which parts of memory are free to hold machine-language routines.

Because of the way ProDOS dynamically allocates and deallocates memory for file buffers, it is somewhat more difficult to manage memory and guarantee which parts of memory will be free to store and protect machine-language routines.

The first sentence is from Apple's book (page 154), the second from *Inside Apple's ProDOS* (page 100), which was published by Reston Publishing Company, a division of Prentice-Hall. The whole chapter is like that. I didn't compare the entire contents of the two books, only this chapter; I don't mean to say there is anything illegal about this type of paraphrasing, because I don't think there is.

The problem is that it's a disservice to both customers and to legitimate writers when publishing companies flood the market with this kind of crap. Compare the two sentences and you'll see Apple's original wording is much clearer. In Prentice-Hall's version, try to figure out what managing memory is more difficult than. It makes you wonder whether the book was even copy edited.

As it happens, both books do an inadequate job of explaining how to install machine-language routines in memory. (See *Softalk*, June 1984, page 159, for an adequate explanation.)

As a book buyer, you should be aware that Prentice-Hall's publishing philosophy is to throw lots of books on the market and see what sticks. Some of them turn out to be quite good, for example, the Quality Software books published by Brady Communcations, also a division of Prentice-Hall (*Understanding the Apple IIe* by Jim Sather and *Beneath Apple ProDOS* by Don Worth and Pieter Lechner). More of them, however, turn out to be mediocre at best, including *Inside Apple's ProDOS*. Buyer beware.

Miscellanea

Adding commands to Basic.system is becoming a good-sized cottage industry. Apple's ProDOS development team put some neat features in Basic.system that allow any number of special routines to be hidden away in memory and called as DOS commands. Lots of stuff that uses this feature is coming out of the woodwork and much of it is quite interesting.

Glen Bredon (321 State Rd, Princeton, NJ 08540), author of the Merlin assembler, has a $20 disk he is selling himself called */PROCMD*. It includes a number of Applesoft editing and debugging utilities, and then some. The disk has a GPLE-like editor; renumber, hold, and merge commands; cross-reference and current value dumps for variables, as well as a value tracer that works while a program is running; copy, sort, type, memory dump, date, and print-using commands; and a double-high resolution graphics package. The documentation is included on the disk. (Incidentally, Glen wrote to say that he documented the changes in IIc Applesoft that we trekked through here last month well over a year ago in the Applesoft source code that comes free with the *Merlin-Pro* assembler—$70 from Roger Wagner Software, P.O Box 582, Santee, CA 92071.)

A routine for stepping through ProDOS directories is part of another interesting disk that appeared in my mailbox. It's called *Developer Disk #1* ($7 from Nite Owl Productions, 5734 Lamar Ave, Mission, KS 66202). This disk has a bunch of routines to assist in the development of programs for the *Night Owl Journal*, a magazine-on-disk scheduled to begin publication this month. The command that steps through directories is called "Quick Index." It allows users to quickly find and select a file even if it's buried inside multiple subdirectories. This is a neat feature we should see more of in ProDOS software. A GETFILE command on this disk allows such selection from within Applesoft programs. The name of the file the user selects is returned in a string variable. Also on this disk are three new INPUT commands. One simply accepts commas and colons; the other two support the underline cursor and delete key, as well as provide GPLE-like string editing functions. One of these is for inputting new strings; the other allows the user to edit existing strings.

The ability to mix routines from various authors is the best part of Basic.system's added commands feature. For example, you can mix Nite Owl's input-anything GETSTR command with Bredon's print USING command to quickly add the two most needed features to Applesoft. Very little memory is used because the commands can be added *independently* of the rest of the commands they come with. Authors of such routines have to know what they're doing, however, to get routines to mix. Several extra steps are required to assure a routine will work with others.

Unfortunately, Apple has never documented what these steps are. (In fact, Apple's own added-command routines, *HELP* and *APA*, which are on the /EXAMPLES disk that comes with *Basic Programming with ProDOS*, don't take these steps and can't be combined with other routines.) The June 1984 **DOStalk** (*Softalk*, page 157), however, did list the required steps and pompously called them the *DOStalk Protocol* for adding machine language utilities to Basic.system. It would be helpful if those of you write these utilities state in your documentation whether you have followed this protocol or not. Refer to the DOStalk article for complete details if you are writing such routines. Here's a summary of the protocol:

```
1. Use Basic.system's GETBUFR call to allocate memory for your routine.
     Be prepared to relocate your routine anywhere; don't expect to be
     the first or only user of GETBUFR.
2. Never use Basic.system's FREEBUFR call; the only crash-proof way to
     free the buffers is to restart Basic.system.
3. When connecting your routine to Basic.system's EXTRNCMD vector or to
     Applesoft's AMPERSAND vector, save what's already in the vector and
     daisy-chain to it if the call is not for you.
```

The **DOStalk** article this originally appeared in included source code for a public domain *TYPE* command. This command displays text files on your screen or printer and demonstrates how to use the protocol. The command's only failing is that it doesn't accept lower-case commands; something all added commands should do. Don't bother to type in the source code from that article, however, the *DOStalk TYPE* command was included on a recent disk distributed by the International Apple Core and consequently is available in user group libraries. The disk is /IAC.44. It also includes a ProDOS version of *File Cabinet*, an early public domain database program for the Apple II.

Ask (or tell) Uncle DOS

Extra 65C02 bulletins

Your old chums at Beagle Bros just put out *Extra K*, a dandy set of programs that let you use the extra 64K in a IIc or 128K IIe. They mention that if you use the program that puts variables into the extra 64K, you need to modify existing machine language routines for sorting arrays and strings. Do you have any tips on how to do this?

I'm told that the 65C02 included in the enhanced ROM upgrade kit will allow my Apple IIe to run cooler and slightly faster. Is this true?

Do you know if the Sider hard drive performs well with bulletin board systems? I plan to put up a board for my department very soon, and would appreciate any input or tips from others who may already be using it for that purpose.

Peter Chin
Brooklyn, N.Y.

Beagle Bros' ace programmers tell me they haven't figured out yet how to get machine language sorts to work with variables stored in auxiliary memory. They point out that if you're not in a hurry, sort routines written in Applesoft will work with Extra K.

I personally think the 65C02 is the most over-hyped item to come down the Apple II pike in some time. It's a nice little chip, and it does run cooler, but if it produced no heat at all it wouldn't significantly affect the temperature inside an Apple II-Plus or IIe because of all the other stuff in there. Bob Sander-Cederlof at Apple Assembly Line *tells me there is one set of infrequently-used machine language instructions that executes in 6 cycles rather than 7, however, there is another set that executes in 7 rather than 6, so there's no net gain there. Programs written to take advantage of some new instructions in the 65C02 could potentially run slightly faster; no programs that I know of, however, other than* SuperCalc 3A *and the IIc firmware, use the new instructions — including the enhanced IIe firmware. Bob also pointed out the chip is available from the manufacturer in versions capable of running faster than the fastest 6502, however, the Apple II is unable to take advantage of this capability without adding a special card.*

The Apple user group in Phoenix had a newsletter article last month about setting up a bulletin board using a Sider and a ProDOS-based bulletin board system called GBBS II *($95 from Micro Data Products, 5739 South Olathe Court, Aurora, Colo. 303-699-1161). The gist of the article was that they were quite happy with the system, however, they admit it took a club member with bulletin board experience 30 to 40 hours to get everything set up as they wanted it. The author of the article was Jerry Cline, who mistakenly let somebody put his phone number in the newsletter: 602-992-7035.*

Short end of joystick

I have been enjoying ***Open-Apple*** since the first issue. I am, however, a little disappointed too. I own (and use a lot!) an Apple II-Plus with shift key modification and a 16K RAM card. Almost everything you print is geared to the IIe or IIc.

I would like to know such things as which of the newer chips I can use in my II-Plus (and tips on installation), what advantages they could give me, and whether or not patches are available so that I can use such software as *AppleWorks* or other new software that was not designed with the II-Plus in mind. I use Applewriter II and Screenwriter. Could I use the new revised *Apple Writer*? Can I functionally do anything to make more than 64K available to the commercial software I use? I still have DOS and am contemplating getting a CP/M card. I'm really beginning to feel left out when I look at the recent Apple software.

I'm reasonably sure that I'm not alone among your readers. Do think about us II-Plus people too. With that off my chest, I should say too that you are the best thing for Apple users since *Softalk* died.

Hannah Lerman
Los Angeles, Calif.

The Apple II-Plus is a classic. A lot of the stuff in ***Open-Apple*** *works on a II-Plus as well as a IIe or IIc — particularly programming information. However, as you point out, a lot of other stuff doesn't.*

There is no advantage to using newer chips or adding memory to **any** *version of the Apple II unless you also have software that will take advantage of the changes. Almost none will so — there's very little to write or talk about in this area other than programming.*

New "productivity" software (word processors, spreadsheets, etc.) is written for the IIe and IIc because they have full keyboards, a standard 80-column screen, and a standard way of adding extra memory. Most of this kind of software, such as newer versions of ***Apple Writer****, won't run on a II-Plus because it has no standards in these areas. On the other hand, a great deal of II-Plus compatible software is being developed and released, but most of it is in other categories — education, games, utilities, and specialized applications.*

The idea of adding CP/M capability to your machine is a good one if your are committed to the II-Plus. Because CP/M was designed as a "universal" operating system to begin with, its programs adapt to various keyboard, memory, and 80-column schemes more easily.

The day will finally come, however, when you will decide to get a newer computer. The ***Open-Apple*** *information that now seems irrelevant to II-Plus users will be much more meaningful then.*

Printing graphics

How can I print out a picture from inside a protected program? I use programs such as *Facemaker* and *Kidwriter* in my classroom and I'd like to be able to print out my students' creations so the kids could take them home and show them off.

Right now I use open-apple/control/reset to get out of these programs and *Grafpak* to print the picture that's left in memory, but the procedure is disruptive and the pictures are full of ugly distortion lines. Help.

S.H. Gidi
Allen Park, Mich.

I use *Apple Business Graphics* on my Apple IIe. It is a Pascal program and several years old. It is configured to print to an Apple Silentype or Qume printer. Although I can get it to print text to my Prowriter with the Grappler card I can't get graphics to print. I've contacted dealers in the past and get the story of expensive drivers I need to purchase. There must be an easier answer, I just haven't found it.

Dave Hamilton
Lincoln, Neb.

Those distortion lines you get in graphics after pressing open-apple/control/reset come from a "feature" of the IIe/IIc Monitor that destroys a couple of bytes on every page of memory as an aid to the dark forces of copy protection.

There is a new kind of printer interface card on the market that solves the problems both of you are having. This kind of card has the power to take control of your computer no matter what kind of program is running — protected, Pascal, or plain — and dump whatever is on your screen (graphics or text or both) to your printer. This kind of card comes with a button you press when you want to print.

One of these new cards is called Print-it! It's available for $199 from Texprint, 220 Reservoir St., Needham Heights, MA 02194 (800-255-1510/617-449-5808). I know of this card only through the company's advertising.

Another is called FingerPrint Plus. It's sold by Thirdware Computer Products, 4747 NW 72nd Ave, Miami, FL 33166 (305-592-7522) for $149. Thirdware sent me one of its cards a couple of months ago to try out. I like it. Its "button" is a small square of cardboard with a cute little fingerprint on it. The button is attached to a thin mylar cable that you can string through the lid crack on the top of your Apple. Stick the pre-gummed button somewhere handy and you're set.

When you press the button, the card grabs control of your machine and displays a menu. From here you can preview all of the Apple text and graphics pages and print any of them. The menu also allows you to change interface card parameters such as margins and line length; allows you to rotate, double, and crop hi-res graphics; allows you to print low-res, double-high-res, mixed text and graphics, and color; allows you to jump into Applesoft or the Monitor; and allows you to choose whether the card's serial port or parallel port will be active (you can also choose both or neither).

The card supports both serial and parallel printers, but comes with only one cable. A second cable is $28. Both cables are over 6 feet long, which is at least a foot longer than the cable that came with my Grappler Plus.

The only reservation I have about the card is that it isn't completely Grappler-compatible. The biggest thing I miss is the Grappler's "NOT SELECTED" message, which appears when your printer isn't turned on or is out of paper or whatever. That little message is worth its weight in IBM software. I've sorely missed it for at least 20 panic-stricken minutes since I installed the Fingerprint Plus — time spent trying to figure out why programs that used to work fine were hanging up. The answer, of course, was as simple as "turn the printer on, stupid", but I'd rather have my interface card tell me that than have to figure it out for myself.

Another esoteric problem I had has to do with the control-I command character. The program that prints ***Open-Apple*** *mailing labels uses printer tabs.*

The control code for tab is control-I. To get control-I to a printer, however, you have to change the interface card command character, which is also control-I, to something else. Otherwise the card will eat them instead of sending them along to the printer (now you know why you couldn't get your printer's tab command to work, Alfie). The standard way to change the command character is to print a control-I control-something-else. The something else then becomes the interface card's command character.

So far so good. But the Grappler Plus reinitializes itself everytime you turn the printer on. Thus you have to do control-I control-something else after every PR#1. The Fingerprint Plus, on the other hand, doesn't reinitialize itself. The second time you turn the printer on it sends your control-I to the printer, swallows your control-something else, and, if the next character you send is a control character (it was an escape in my case) changes the command character a second time. Now you have a mess on your hands, as well as a puzzle.

If you are writing your own programs, of course, all this is fixable, once you figure out what it going on. But if you are using commercial software that's Grappler-compatible, you could end up with some problems. (Incidentally, the May tip from Bernard Goodman on getting a hex dump of what's going to the printer (page 39) was invaluable in solving this puzzle.)

The manual that comes with the Fingerprint Plus is pretty good for printer-related documentation, which is notoriously bad, but not nearly detailed enough for a perfect world. In particular, it doesn't warn you in big letters on every page not to push the button while a disk drive is saving something. In fact, it doesn't warn you about this at all — even though it's an easy way to destroy the data on disks.

All in all, however, the advantages of the Fingerprint Plus compared to the Grappler Plus are quite explicit, while the disadvantages are esoteric. The Fingerprint Plus not only allows you to print from within any program, but it has several graphics-printing features the Grappler Plus lacks. Look at the price, too — on a features-per-dollar basis the Fingerprint Plus looks to me like an exceptional card.

Both sides win again

I just wanted to add my two cents to the controversy over using the back side of single-sided disks. The disk manufacturers will tell you that only one side is certified; but in fact, since some drives write on the back of the disk and others on the front, they **don't know which side your disk drive is writing to**. Therefore they have to certify both sides; witness the fact that there are "data holes" on both sides of the disk.

There's a potential problem of dirt accumulating in the dust jacket and then being released when the disk spins the "other" way, but I've yet to see data loss due to this in my six years of flipping floppies.

Keep up the great work with ***Open-Apple***; next to ***Nibble***, it's my favorite Apple reading.

David Szetela
Concord, Mass.

*I hate to reprint letters in which people say they prefer **Nibble** to **Open-Apple**, but after last month's smart-aleck exclamation point expose (page 45), I owe them one. Just so you understand where this correspondent is coming from, he's **Nibble's** Managing Editor.*

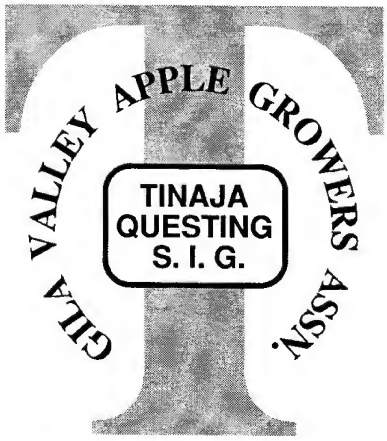
Edasm problems

Having used the mini-assembler to start learning machine language, I purchased the ProDOS Editor-Assembler, mostly because of its well-written tutorial. However, since there is no provision for multiple ORGs in the program section of the source code, I haven't been able to enter even short data tables at absolute memory locations without using the Monitor or another file. Is there any technique around this limitation?

Stephen Gale
Keyport, N.J.

*According to Don Lancaster's Assembly Cookbook for the Apple II/IIe ($21.95 from Howard W Sams & Co, 4300 West 62nd St, Indianapolis, Ind. 46268), you **can** use multiple ORGs with EDASM. This book is based on EDASM — it's a good introduction to using assembly language. Apple gave me a copy of EDASM at a ProDOS seminar I went to a couple of years ago and it was so different from the other assemblers I've used that I immediately filed it away and haven't seen it since, so I don't know much about it. You might try calling or writing Lancaster, in care of Synergetics, Box 809, Thatcher, Ariz. 85552 602-428-4073.*

Speaking of Lancaster, he sent the following logo this month with the caption "This is a Laserwriter image done with Applewriter on an Apple IIe. Ask your local printer to estimate the typesetting cost on this for you. Be sure to wear good running shoes; you will need them." The image has been reduced 50 per cent.



BRUN bug/More on RGB

I am perplexed by something from the May issue of ***Open-Apple***. Why do we have to switch SNGL.RES on and off to make double hi-res work on an RGB screen? Is this a timing problem, or what?

I'm wondering whether this is related to a problem I'm having with a double hi-res screen dump program I wrote. When I BRUN the program it hangs, but if I BLOAD and CALL it instead, it runs just fine. Here's a small test program that demonstrates the bug. It seems to hang when it reaches the main mem/aux mem flip:

```
0300:8D 5E C0   STA SNGL.RES.OFF  (double-res on)
0303:8D 0D C0   STA COL80.ON
0306:8D 50 C0   STA GRAPH.ON
0309:8D 57 C0   STA HIRES.ON
030C:8D 53 C0   STA MIXGR.ON
030F:8D 01 C0   STA STORE80
0312:8D 55 C0   STA PAGE2.ON
0315:AD 00 20   LDA $2000
0318:20 DA FD   JSR PRBYTE
031B:8D 54 C0   STA PAGE1.ON
031E:AD 00 20   LDA $2000
0321:20 DA FD   JSR PRBYTE
0324:60         RTS
```

Alain Toutant
Ste-Angele-de-Laval, Quebec

Your problem is actually a bug in the DOS 3.3 BRUN command. The program doesn't hang until it gets to the RTS.

When you BRUN a program, DOS loads your code and does a JMP to it from within a subroutine. The RTS at the end of your code is taken as the end of that DOS subroutine.

Any program running under DOS 3.3 that is BRUN should save the value Uncle DOS has stored at byte 43609 ($AA59). This value is critical to your program's finding its way home again, but it gets overwritten when DOS intercepts a character passing through the I/O hooks. If you save $AA59 before doing any input/output, and replace it before your final RTS, your problems will go away. Look at lines $408B and $409A of March's INCINERATOR (page 18) for an example.

Poking the SNGL.RES and 80COL switches as shown in the May issue (page 36) shifts bits into two flags on the RGB card, which in turn tell the card which video mode to display. In addition to the three modes discussed earlier (560 x 192 monochrome, 140 x 192 16-color, and a mix of 560 and 140), it turns out there is also a fourth mode — 160 x 192 in 16 colors. This mode uses all 8 bits of each byte (two 4-bit pixels per byte), rather than 7 as the other modes do. Consequently, it is much easier to program. The first two pixels of line one are stored at $2000 in auxiliary memory, the next two pixels are in main memory, and so on across the line.

Here's the correct sequence for poking the switches to get any of the four modes (it doesn't matter what value you poke). The 80COL off and on switches are at 49164 and 49165 ($C00C-C00D) and the SNGL.RES off and on switches are at 49246 and 49247 ($C05E-C05F). (Note that SNGL.RES off is the same thing as DBL.RES on, which is what I wish I had called this pair of switches last time. They are also known as AN3, because they also control an output on the game port called annunciator #3.):

RGB Control Sequences

560 x 192 monochrome	140 x 192 16-color	560 & 140 mixed	160 x 192 16-color
80C off	80C on	80C off	80C on
DBL on	DBL on	DBL on	DBL on
DBL off	DBL off	DBL off	DBL off
		80C on	80C off
DBL on	DBL on	DBL on	DBL on
DBL off	DBL off	DBL off	DBL off
80C on			80C on
DBL on	DBL on	DBL on	DBL on

All the above modes require that both DBL.RES and 80COL end up turned on.

BEWARE, programmers, of leaving DBL.RES on and switching to 40-column text. Users with RGB cards get a color text display in this situation that is often unreadable if turned on by accident. With DBL.RES on, 40-column text on RGB-equipped Apples appears in combinations of 16 foreground-background colors. The display page in main memory holds the text and the display page in auxiliary memory holds color information. The first nibble of each color byte holds the color of the text (the foreground), the

second nibble holds the background color. If both nibbles hold the same value, the text blends into the background and becomes invisible.

Token tinkering

In your June issue (page 42), you tell how the IIc cassette commands point to the same location as the & command. There is a simpler method than the one you give to distinguish between the commands. When Applesoft calls a command routine, the Y register contains an index to the command. The following three instructions will move the index into the A register and convert it into the command token ($9A=SHLOAD, $A7=RECALL, and so on, as listed in the June issue):

```
TYA
SEC
ROR
```

Glenn Chappell
Overland Park, Kans.

Here's a little program that's demonstrates Chappell's technique:

```
100 REM poke $300 in page-3 ampersand vector
110 C$="3F6:00 03"
120 GOSUB 500

130 REM poke Chappell's routine at $300
140 C$="300:9B 3B 6A 4C DA FD"
150 GOSUB 500

160 END

500 C$=C$+" N D9C6G" : REM S.H. Lam routine
510 FOR I=1 TO LEN(C$)
512 : POKE 511+I, ASC(MID$(C$,I,1))+128
514 NEXT
520 POKE 72,0 : CALL -144
530 RETURN
```

After running this program, enter the 6 possible IIc ampersand commands (SHLOAD, RECALL, STORE, LOAD, SAVE, &) directly on a IIc keyboard. The program causes each to print its associated token on the screen.

Make byte low

I am finally gritting my teeth and attempting a wholesale conversion from DOS 3.3 to ProDOS. One of the most heavily used word processors in our company is *Format II Enhanced*, because of its easy on-page orientation and the ability to do columns for script writing.

However, the ProDOS CONVERT utility inexplicably sets the high bit on all the text files coverted over. *Format* uses the high bit, so converted files are a mess. Several phone calls to Kensington Microware resulted in a vague commitment to work on it sometime and Apple referred me to my local dealer (Ho, Ho). Is there a way to fix CONVERT? Or is there a nice simple little program that can take a converted file and turn off the high bit in every byte?

By the way, I haven't noticed any mention of it in ***Open-Apple***, but one of the key reasons for my switch to the combination of ProDOS and the IIe enhancement is the wonderful *SuperCalc 3A*. This is one of the best indications yet of the increasingly long life of the II series.

Flip Baldwin
Salinas, Calif.

Take the basic Basic.copy program from the front part of this issue. Add the S.H. Lam routine near the beginning of the program (see the February issue, page 12, for details on this; the answer to the last letter includes the actual routine) to load the following machine language stuff at $300:

```
C$="300:1B AD D7  BE B5 3C  6D DB BE
      B5 3E AD  DB BE B5  3D 6D DC
      BE B5 3F  A0 00 B1  3C 29 7F
      91 3C 20  BA FC 90  F5 60"
```

Now add a CALL 768 to the copy program between the BLOAD and the BSAVE. The program will then load a file and call the machine lanugage subroutine at 768 ($300). That routine zips through the image of the file in memory and turns off all the high bits. When it's done, the main program will resave the file.

ProDOS 40-track bug

In the April ***Open-Apple*** you published some patches to make ProDOS 1.1.1 and FILER 1.1 work with 40-track drives. You point out that the patches are based on Worth and Lechner's patches in *Beneath Apple ProDOS*. There seems to be an error in both. The change of $23 (35) to $28 (40) makes sense but shouldn't $18 ($0118 = 280) be changed to $40 ($0140 = 320) instead of $39 ($0139 = 313)?

Worth and Lechner claim the patches will allow for 320 blocks instead of 280. I have formatted disks both ways and inspected them with a disk zap utility, and it seems to me that using $28 and $39 formats the disk for 40 tracks but only allows for 313 of 320 blocks to be used. Am I right?

A different question: you note parenthetically that block 2 = track 0, sector 11; but Worth and Lechner's Table 3.1 shows block 2 in sectors 8 and 10. Who's right?

George Tylutki
La Plume, Penn.

You're right about the patches in our April issue (page 32). Where you see $39 change it to $40. Pencil this change in your back issue now, everybody.

The confusing thing about Worth and Lechner's Table 3.1 in Beneath Apple ProDOS *is that it relates ProDOS blocks to* ***physical*** *disk sectors. The physical sectors on DOS 3.3 and ProDOS disks are identical; a disk zap utility based on either operating system can access the other's disks. But Worth and Lechner's table is meaningless to anyone trying to examine a ProDOS block with a DOS 3.3 zap utility because DOS 3.3 sector numbers are* ***logical*** *numbers; a little table inside DOS is used to change them to physical sector numbers just before accessing a disk.*

The shaded box on this page has a table similar to Worth and Lechner's that converts ProDOS blocks to DOS 3.3 tracks and sectors. (Note that since ProDOS blocks are twice as big as a DOS 3.3 sector, it takes two sectors to make one block.)

ProDOS-DOS 3.3 disk sector conversion tables

DECIMAL

track	DOS 3.3 sector numbers 0+14	13+12	11+10	9+8	7+6	5+4	3+2	1+15
	ProDOS block numbers							
0 :	0	1	2	3	4	5	6	7
1 :	8	9	10	11	12	13	14	15
2 :	16	17	18	19	20	21	22	23
3 :	24	25	26	27	28	29	30	31
4 :	32	33	34	35	36	37	38	39
5 :	40	41	42	43	44	45	46	47
6 :	48	49	50	51	52	53	54	55
7 :	56	57	58	59	60	61	62	63
8 :	64	65	66	67	68	69	70	71
9 :	72	73	74	75	76	77	78	79
10 :	80	81	82	83	84	85	86	87
11 :	88	89	90	91	92	93	94	95
12 :	96	97	98	99	100	101	102	103
13 :	104	105	106	107	108	109	110	111
14 :	112	113	114	115	116	117	118	119
15 :	120	121	122	123	124	125	126	127
16 :	128	129	130	131	132	133	134	135
17 :	136	137	138	139	140	141	142	143
18 :	144	145	146	147	148	149	150	151
19 :	152	153	154	155	156	157	158	159
20 :	160	161	162	163	164	165	166	167
21 :	168	169	170	171	172	173	174	175
22 :	176	177	178	179	180	181	182	183
23 :	184	185	186	187	188	189	190	191
24 :	192	193	194	195	196	197	198	199
25 :	200	201	202	203	204	205	206	207
26 :	208	209	210	211	212	213	214	215
27 :	216	217	218	219	220	221	222	223
28 :	224	225	226	227	228	229	230	231
29 :	232	233	234	235	236	237	238	239
30 :	240	241	242	243	244	245	246	247
31 :	248	249	250	251	252	253	254	255
32 :	256	257	258	259	260	261	262	263
33 :	264	265	266	267	268	269	270	271
34 :	272	273	274	275	276	277	278	279
35 :	280	281	282	283	284	285	286	287
36 :	288	289	290	291	292	293	294	295
37 :	296	297	298	299	300	301	302	303
38 :	304	305	306	307	308	309	310	311
39 :	312	313	314	315	316	317	318	319

HEX

track	DOS 3.3 sector numbers 0+E	D+C	B+A	9+8	7+6	5+4	3+2	1+F
	ProDOS block numbers							
0 :	0	1	2	3	4	5	6	7
1 :	8	9	A	B	C	D	E	F
2 :	10	11	12	13	14	15	16	17
3 :	18	19	1A	1B	1C	1D	1E	1F
4 :	20	21	22	23	24	25	26	27
5 :	28	29	2A	2B	2C	2D	2E	2F
6 :	30	31	32	33	34	35	36	37
7 :	38	39	3A	3B	3C	3D	3E	3F
8 :	40	41	42	43	44	45	46	47
9 :	48	49	4A	4B	4C	4D	4E	4F
A :	50	51	52	53	54	55	56	57
B :	58	59	5A	5B	5C	5D	5E	5F
C :	60	61	62	63	64	65	66	67
D :	68	69	6A	6B	6C	6D	6E	6F
E :	70	71	72	73	74	75	76	77
F :	78	79	7A	7B	7C	7D	7E	7F
10 :	80	81	82	83	84	85	86	87
11 :	88	89	8A	8B	8C	8D	8E	8F
12 :	90	91	92	93	94	95	96	97
13 :	98	99	9A	9B	9C	9D	9E	9F
14 :	A0	A1	A2	A3	A4	A5	A6	A7
15 :	A8	A9	AA	AB	AC	AD	AE	AF
16 :	B0	B1	B2	B3	B4	B5	B6	B7
17 :	B8	B9	BA	BB	BC	BD	BE	BF
18 :	C0	C1	C2	C3	C4	C5	C6	C7
19 :	C8	C9	CA	CB	CC	CD	CE	CF
1A :	D0	D1	D2	D3	D4	D5	D6	D7
1B :	D8	D9	DA	DB	DC	DD	DE	DF
1C :	E0	E1	E2	E3	E4	E5	E6	E7
1D :	E8	E9	EA	EB	EC	ED	EE	EF
1E :	F0	F1	F2	F3	F4	F5	F6	F7
1F :	F8	F9	FA	FB	FC	FD	FE	FF
20 :	100	101	102	103	104	105	106	107
21 :	108	109	10A	10B	10C	10D	10E	10F
22 :	110	111	112	113	114	115	116	117
23 :	118	119	11A	11B	11C	11D	11E	11F
24 :	120	121	122	123	124	125	126	127
25 :	128	129	12A	12B	12C	12D	12E	12F
26 :	130	131	132	133	134	135	136	137
27 :	138	139	13A	13B	13C	13D	13E	13F

Basic.system trace bug

Both versions of Basic.system contain an ephemeral but interesting bug that suddenly triggers trace mode and just as suddenly disappears.

Consider the following 4-liner by J.R. Wakefield from the "Letters" section of the April 1985 ***Nibble***:

```
10 GET A$: PRINT
20 IF A$="A" THEN FLASH : PRINT "ERROR" : GOTO 40
30 FLASH : PRINT "OK"
40 NORMAL : GOTO 10
```

If an upper case "A" is pressed, tracing begins. All other input produces a normal response. Abnormal tracing ceases when another text mode command is encountered.

Tracking the bug to its lair took considerable effort and will be the subject of *Disassembly Lines* column in the December 1985 or January 1986 ***Nibble***. For now, here's a summary of the problem and a recommended solution.

Unlike DOS 3.3, Basic.system bullies the Applesoft new statement and trace handler (NEWSTT, $D7D2-$DB56) into kicking trace information back to Basic.system. Basic.system uses the trace information as a signal that a new command is being executed. It checks out the token of the command for screening and pre-processing. It does all this by making sure that the Applesoft trace flag (TRCFLG, $F2) is always on (high bit set). The true status of trace mode is kept within Basic.system's global page (DTRACE, $BE41). By checking this byte Basic.system determines whether to pass the tracing information on to the screen or not.

Basic.system also stores an image of the normal trace character, which happens to be "#" (ASCII $A3). Whenever anything is printed, Basic.system, like DOS 3.3, intercepts it. Basic.system, however, checks to see if the source of the information is Applesoft's trace mode by first checking if the pound sign is being output, and if so, by determining whether the output originates from NEWSTT.

After Applesoft processes a FLASH command, trace's leading pound sign acquires an ASCII value of $E3 instead of $A3. Thus, when Basic.system sees a FLASH token, it changes the trace character image from $A3 to $E3. But, unfortunately, not all contingencies are taken into account.

When Applesoft executes an IF statement, trace data is sent to Basic.system as usual. However, if the formula following the IF statement is true, Applesoft will execute the command following THEN *without* outputting any additional trace information. Thus, if FLASH is executed inside an IF-THEN statement, Basic.system never sees the FLASH token and doesn't change the trace character image from $A3 to $E3. Consequently, Basic.system doesn't recognize the following flashing trace data for what it is and sends it on to the screen.

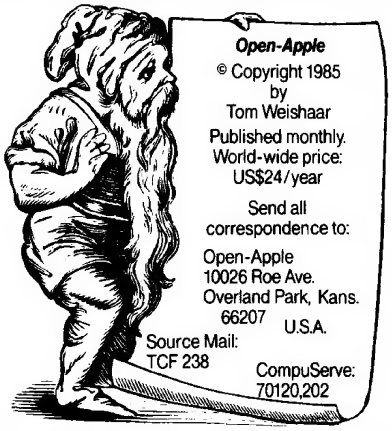
The dynamics of this adverse interaction between Applesoft and Basic.system is complex and illustrates that debugging involves more than the mere examination of code segments.

Fixing the bug is as simple as realizing that the concept of a trace character image is unnecessary. If output comes from NEWSTT, the pound sign **must** be the character being processed. With this in mind, here is my patch for Basic.system 1.1 (for 1.0 change the starting address to $9E5B):

```
9E2C:48 8E 3F BE
9E30:8A BD 04 10 C9 12 D0 0A
9E38:BD 05 01 C9 DB D0 03 68
9E40:80 74 68 EA EA EA
```

I stress that this fix is no substitute for revising Basic.system's output handler. If someone would rather avoid the bug than correct it (a mortal sin for hackers), just be certain that Applesoft commands pre-processed by Basic.system (e.g. TRACE, NOTRACE, NORMAL, INVERSE, FLASH, RESUME) do not immediately follow a THEN.

Sandy Mossberg
Rye Brook, NY



Bugs and the future

Here is an answer for "Problem too complex" (February, page 15). A="A" internally uses a zero page temporary string descriptor for the literal "A." This descriptor is freed upon successful completion of the statement. However, the statement does not complete because it is in error. ONERR redirects the program to execute the statement again. Soon the limited number of internal string descriptors is exhausted. FORMULA TOO COMPLEX is generated when no more are available. A$=A doesn't use an internal zero page temporary sting descriptor. Too many internal nested string generations will also produce the same result.

With regard to "Interrupts and the Sun" (June, page 46), ProDOS *does* use $45! The *ProDOS Technical Reference Manual* (page 110-111) describes the locations to be used by intelligent controller boards, such as the one used with the ProFile. The ProFile may be called directly with the following memory locations set:

```
$42       command (0=status, 1=read, 2=write)
$43       unit number (dsssxxxx, d=drive, sss=slot)
$44-$45   buffer pointer (512 byte area)
$46-$47   block number
```

I have written routines to directly access the ProFile using this layout and JSR $CsXX, where s is the slot number and XX is the value at $CsFF. Notice that the last 5 of the 6 bytes are the same as the ProDOS Read_Block ($80) and Write_Block ($81) MLI calls. The error codes returned by the ProFile ROM in the A-reg are exactly the same as the ProDOS MLI codes ($27=I/O ERROR, $28=NO DEVICE CONNECTED, $2B=WRITE PROTECTED), with carry set. So, yes, $45 is used by ProDOS. By the way, you can see why DOS 3.3 doesn't support the ProFile, since the I/O is done in units of 512 byte blocks.

So why won't DOS 3.3 be supported on future versions of the Apple II? Well, we all complained that DOS 3.3 didn't have enough features, had enough bugs, and could not easily handle a number of significant new devices (hard disks, interrupt generators, etc.). Everyone and his uncle had a patch to DOS 3.3 for something or other. And Apple Computer listened. Thus, along came ProDOS. The amazing part is that the Applesoft interface is very nearly the same (and Applesoft is even the same; I converted *Softgraph* from ***Softalk*** with less than a half dozen changes). In a nutshell, ProDOS has all of the capabilities that we wanted in DOS 3.3 (but really could not be put in DOS 3.3), fixes many problems (how many years to fix append), and is expandable for the future.

I believe the potential for DOS 3.3 is far too limited for the future of the Apple II computer. The DOS 3.3 append error was almost unfixable. How many bugs still exist that will not be found until someone attempts to use the code on a device that was not originally supported (like a 3-1/2 inch Sony floppy with variable speed depending on track number)?

If there is a future for the Apple II, and if that future includes faster CPU chips (65816), more memory, larger disk devices, and extended human interface capabilities (mouse and graphics), then DOS 3.3 just does not make it. ProDOS is a better base.

With the departure of the Apple III and SOS, ProDOS is the only good operating system left at Apple Computer. The Mac file handling has been the pits. It is even worse than the days of early DOS 3 (at least, that is what I read, and of course, only the bad stuff gets printed). I used to have all kinds of problems with DOS 3.3 (it had to be babied). I don't have any problems with ProDOS. That means a lot when I have thousands of hours of work invested on my ProFile. Not one bad block. Not one lost track. Not one I/O error. And the Apple Mouse worked right away. And the extended memory worked right away. And the IIe enhancement worked right away. Maybe I am spoiled. When I buy something, I expect it to work. All of my ProDOS products work. Can this be said about DOS 3.3? What kind of changes need to be made to DOS 3.3 to support the 65816?

Ken Kashmarek
Eldridge, Iowa

Thanks for your help with FORMULA TOO COMPLEX and $45. I suspect interrupts are disabled when ProDOS calls intelligent controller boards and that's why this call is allowed to use $45. Direct calls such as yours should also either disable interrupts or plan not to use them on II-Pluses and original IIes.

I agree with your comments on ProDOS. As hard disks and other high-capacity storage devices come down in price most other people will agree with you, too. However, you have to admit that the biggest problem with ProDOS and Basic.system right now is that we just don't know what bugs lurk within them. There certainly appear to be a few — as is expected in any major piece of software.

Yet healthy enthusiasm for ProDOS is no reason to abandon all the effort that's gone into learning about DOS 3.3. The problem is not that DOS 3.3 doesn't support the ProFile, but that the ProFile — alone among hard disks for the Apple II family — doesn't support DOS 3.3. The technical problems are solvable; Apple in its arrogance toward the market simply decided not to solve them. But compare ProFile sales to those of other hard disks for the II and you'll see that Apple, in this case at least, is paying for its disregard of the customer.

DOS 3.3 has bugs, sure, but why, just when we finally know what they are and have thousands of people who know how to program around them, should we dump it? It's a perfectly good operating system for many, many floppy-based applications. I continue to insist that any future Apple that can't boot DOS 3.3 is just an Apple, too; it's not a member of the Apple II family.